OptiInfra E2E System Tests

PART 2: Execution & Validation Guide

Version: 1.0
Phase: 5.8 - E2E System Tests
Last Updated: October 28, 2025

Table of Contents

1. Overview

Purpose

This document provides step-by-step instructions for executing the OptiInfra E2E System Tests and validating the results. It complements PART 1 (Implementation) which contains the test code.

What Gets Tested

• Complete multi-agent workflows (Cost, Performance, Resource, Application)
• Orchestrator coordination and conflict resolution
• Customer portal integration and real-time updates
• End-to-end optimization execution (recommendation → approval → execution → validation)
• Data persistence across all databases (PostgreSQL, ClickHouse, Qdrant, Redis)
• Security and authentication mechanisms
• Error handling and rollback capabilities

Test Duration

Test Type	Duration	Count
E2E Complete Workflows	5-10 min each	8 scenarios
Integration Tests	1-2 min each	20 tests
Performance Tests	2-5 min each	5 tests
Security Tests	30 sec each	10 tests
Total Suite	60-90 minutes	43 tests

2. Prerequisites

Required Software

Software	Version	Purpose
Docker	20.10+	Container runtime
Docker Compose	2.0+	Orchestration
Python	3.11+	Test execution
pytest	7.0+	Test framework
Git	2.30+	Code repository
Make	4.0+	Build automation

Python Dependencies

Install all dependencies:

```
pip install -r requirements-test.txt
```

Required Services

✓ All OptiInfra services built (orchestrator, agents, portal)
✓ Docker images available locally or in registry
✓ Sufficient system resources (8GB RAM, 20GB disk minimum)

■■ **IMPORTANT:** Run tests in isolated environment only. Never run against production!

3. Environment Setup

Step 1: Clone Repository

```
git clone https://github.com/optiinfra/optiinfra.git
cd optiinfra
```

Step 2: Verify Prerequisites

```
make verify-prerequisites
```

This checks all required software is installed.

Step 3: Build Docker Images

```
make build-all
```

Expected output:

```
Building orchestrator... ✓
Building cost-agent... ✓
Building performance-agent... ✓
Building resource-agent... ✓
Building application-agent... ✓
Building portal... ✓
```

Step 4: Start Test Environment

```
make start-test-env
```

This starts:
• PostgreSQL (port 5433)
• ClickHouse (port 8124)
• Redis (port 6380)
• Qdrant (port 6334)
• LocalStack (mock AWS, port 4567)
• Orchestrator (port 8001)
• All 4 agents
• Customer Portal (port 3001)

Step 5: Verify Services Health

```
make health-check
```

Expected: All services report healthy status.

■ **Environment Ready!** Proceed to test execution.

4. Execution Workflow

Quick Start (All Tests)

To run the complete test suite:

```
make test
```

Selective Test Execution

E2E tests only:

```
make test-e2e
```

Integration tests:

```
make test-integration
```

Performance tests:

```
make test-performance
```

Security tests:

```
make test-security
```

Fast tests (skip slow):

```
make test-fast
```

Specific test file:

```
pytest tests/e2e/test_spot_migration.py -v
```

Single test function:

```
pytest tests/e2e/test_spot_migration.py::test_complete_spot_migration_workflow
-v
```

Test Execution Flow

1. **Environment Start:** Docker Compose spins up all services
2. **Initialization:** Pytest loads fixtures and test data
3. **Test Execution:** Tests run sequentially (E2E) or parallel (unit/integration)
4. **Assertion Validation:** Results verified against expected outcomes
5. **Cleanup:** Test data removed, services reset
6. **Reporting:** Coverage and results generated

5. Test Scenarios Execution

This section details each E2E test scenario with expected outcomes.

Scenario 1: Spot Instance Migration

File: test_spot_migration.py
Duration: ~8 minutes
What it tests:
• Cost agent detects optimization opportunity
• Multi-agent validation (Performance + Application agents)
• Customer approval workflow
• Blue-green deployment execution
• Cost reduction validation (>40%)
• Quality maintained (>95%)
• Learning loop stores success pattern

Expected Output:

```
■ PHASE 1: Recording initial state...
Initial monthly cost: $120,000
■ PHASE 2: Triggering cost agent analysis...
Analysis ID: anal_abc123
■ PHASE 3: Waiting for recommendation...
■ Recommendation generated
Estimated savings: $18,000/month
...
■ SPOT MIGRATION E2E TEST PASSED
```

Scenario 2: Performance Optimization

File: test_performance_optimization.py
Duration: ~7 minutes
Tests: KV cache tuning, quantization, latency improvement (2-3x)

Scenario 3: Multi-Agent Coordination

File: test_multi_agent_coordination.py
Duration: ~6 minutes
Tests: Conflict detection and resolution between agents

Scenario 4: Quality Validation

File: test_quality_validation.py
Duration: ~5 minutes
Tests: Application agent detects quality degradation and triggers rollback

Scenario 5: Complete Customer Journey

File: test_complete_customer_journey.py

Duration: ~10 minutes (longest)
Tests: Signup → Infrastructure onboarding → Agent deployment → First savings

Scenario 6: Rollback

File: test_rollback_scenario.py | **Duration:** 4 min
Tests: Automatic rollback on failures

Scenario 7: Conflict Resolution

File: test_conflict_resolution.py | **Duration:** 6 min
Tests: Orchestrator priority-based decisions

Scenario 8: Cross-Cloud

File: test_cross_cloud_optimization.py | **Duration:** 8 min
Tests: Multi-cloud resource optimization

6. Results Interpretation

Understanding Test Output

Pytest provides detailed output for each test:

Success Indicators:
✓ Green checkmarks for passed tests
✓ 'PASSED' status for each test function
✓ No assertion errors in output
✓ All expected phases completed
✓ Coverage report generated

Failure Indicators:
✗ Red X for failed tests
✗ 'FAILED' status with error traceback
✗ AssertionError with expected vs actual values
✗ TimeoutError if test exceeds duration
✗ Connection errors to services

Coverage Report

After test completion, view coverage:

```
open htmlcov/index.html
```

Component	Target	Acceptable
Orchestrator	90%	85%
Cost Agent	85%	80%
Performance Agent	85%	80%
Resource Agent	85%	80%
Application Agent	85%	80%
Portal API	80%	75%
Overall System	85%	80%

7. Validation Criteria

Tests must meet these criteria to pass:

Category	Criteria	Validation Method
Functional	All workflows complete successfully	Assert optimization.status == 'completed'
Performance	Optimizations execute within time limits	Duration < timeout threshold
Cost	Actual savings ≥ 80% of predicted	assert_cost_reduced()
Quality	No quality degradation > 5%	assert_quality_maintained()
Security	Unauthorized access blocked	Response status == 401/403
Multi-Agent	Agents coordinate properly	assert_multi_agent_coordination()
Rollback	Failed optimizations rollback	Original state restored
Data Integrity	All databases consistent	Cross-DB foreign key checks

Pass Criteria Summary

✓ All 8 E2E scenarios pass
✓ All 20 integration tests pass
✓ All 5 performance tests pass
✓ All 10 security tests pass
✓ Overall test coverage ≥ 80%
✓ No critical bugs found
✓ Test execution time < 90 minutes

■ All criteria must be met for release approval.

8. Troubleshooting

Common Issues and Solutions

Issue: Services not starting

```
Solution:
• Check Docker is running: docker ps
• Check port conflicts: netstat -an | grep LISTEN
• Restart Docker: systemctl restart docker
• Clean environment: make clean && make start-test-env
```

Issue: Tests timing out

```
Solution:
• Increase timeout in conftest.py
• Check service logs: docker-compose logs -f
• Verify network connectivity between containers
• Check system resources (RAM, CPU)
```

Issue: Database connection errors

```
Solution:
• Wait longer for DB initialization (10-15 seconds)
• Check DB containers: docker-compose ps
• Verify connection strings in conftest.py
• Reset databases: make clean && make test
```

Issue: Tests fail intermittently

```
Solution:
• Check for race conditions in test code
• Increase polling intervals
• Ensure proper cleanup between tests
• Run tests in isolation: pytest tests/e2e/test_name.py -v
```

Issue: Low coverage

```
Solution:
• Check which files are excluded in .coveragerc
• Add missing test scenarios
• Review untested code paths
• Use pytest --cov-report=html for detailed analysis
```

Getting Help

If issues persist:
1. Check logs: docker-compose -f docker-compose.e2e.yml logs
2. Review pytest output with -vv flag for verbose details
3. Contact DevOps team with log output
4. File bug report with reproduction steps

9. CI/CD Integration

GitHub Actions Integration

E2E tests run automatically on:
• Pull requests to main branch
• Daily scheduled runs (nightly builds)
• Manual workflow dispatch
• Pre-release tags

Workflow Configuration

File: .github/workflows/e2e-tests.yml

```
name: E2E Tests
on: [pull_request, schedule]
jobs:
test:
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v3
- name: Run E2E Tests
run: make test
- name: Upload Coverage
uses: codecov/codecov-action@v3
```

Test Failure Handling

When tests fail in CI:
1. CI pipeline stops (blocks PR merge)
2. Slack notification sent to #eng-alerts
3. Developer reviews failure logs
4. Fix applied and tests re-run
5. PR can merge only after tests pass

■■ **Never merge failing tests!** All E2E tests must pass before production deployment.

10. Reporting

Test Reports Generated

JUnit XML Report
Location: `test-results.xml`
Purpose: For CI/CD integration

HTML Coverage Report
Location: `htmlcov/index.html`
Purpose: Detailed line-by-line coverage

Terminal Coverage
Location: `console output`
Purpose: Quick coverage summary

Test Duration Report
Location: `pytest output`
Purpose: Time taken per test

Failure Screenshots
Location: `screenshots/`
Purpose: For failed UI tests (if applicable)

Sample Test Report

```
==================== test session starts ====================
platform linux -- Python 3.11.5, pytest-7.4.3
collected 43 items

tests/e2e/test_spot_migration.py::test_complete... PASSED [8%]
tests/e2e/test_performance_optimization.py::... PASSED [16%]
tests/e2e/test_multi_agent_coordination.py::... PASSED [24%]
...
==================== 43 passed in 78.34s ====================

Coverage Summary:
orchestrator/ 87%
agents/cost_agent/ 86%
agents/perf_agent/ 84%
agents/resource_agent/ 85%
agents/app_agent/ 83%
portal/ 81%
------------------------------------------------------------
TOTAL 85%
```

■ **Test suite complete! All validations passed.**

Appendix: Quick Reference

Essential Commands

Command	Description
make test	Run all tests
make test-e2e	Run E2E tests only
make test-fast	Run fast tests (skip slow E2E)
make clean	Clean up test environment
make health-check	Check services health
pytest -v	Verbose test output
pytest -x	Stop on first failure
pytest -k 'spot'	Run tests matching 'spot'
pytest --lf	Run last failed tests

Support Contacts

DevOps Team: devops@optiinfra.com
QA Lead: qa-lead@optiinfra.com
Slack Channel: #eng-testing
Documentation: https://docs.optiinfra.com/testing

--- End of Document ---